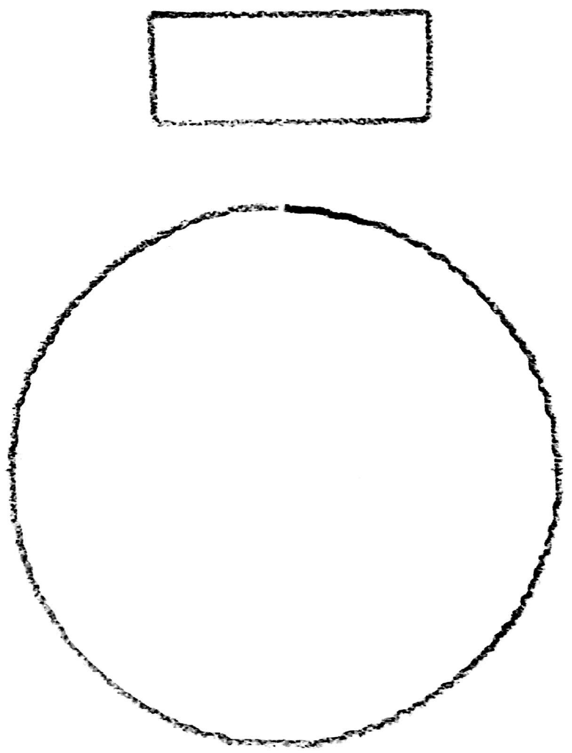
THE UNIVERSITY OF AUCKLAND NEW ZEALAND

2D Controlled Marking Board


Rectangle and Circle Produced During Lab Demonstration

Naseeb Khan & Daniel McGinn

MECHENG 312

11 May 2017

Contents

0.1 Slide Rail Dynamics and Interfacing

0.1.1 Motor Voltage Control

A basic virtual instrument was created during the labs to control voltages sent to each of the motors. Only running motor one in the positive direction caused the pen to move diagonally right and down. Similarly, running the second motor in the positive direction caused the pen to move diagonally right and up. By varying the magnitude of the voltage sent to each motor, the XY platform could be controlled based on the principles of vector addition. For example, if the same positive voltage is sent to both motors, the pen will move horizontally to the right. The 8 cardinal directions can be achieved by multiplying the voltage by 1, -1, or 0. The necessary multiplication factors to achieve motion in each direction is summarised in the table below.

Direction of Movement	Motor 1 Voltage	Motor 2 Voltage
Left	-	-
Right	+	+
Up	-	+
Down	+	-
Left & Up	-	0
Left & Down	0	-
Right & Up	0	+
Right & Down	+	0

Table 1: Directions and their Corresponding Motor Voltages

0.1.2 Interfacing

'DAQmx Create Virtual Channel' VIs were used to read the physical channel inputs from the limit switches. A 'DAQmx Read' VI returns a Boolean variable that indicates whether the limit switch is pressed. This is used to prevent the motors from running the slide rail into the sides of the XY platform. Similarly, virtual channels were created from the two physical channels connected to the motors. 'DAQmx Write' VIs are utilised to produce analog voltages to control the two motors. 'DAQ CI Cnt Edges' are utilised to count the encoders, 'DAQmx Start Task' is used so that the reading of the encoders is continuous.

0.2 Control Program

0.2.1 Overview of the Control Program

The control program we have created behaves by generating co-ordinates of shapes and reaching those co-ordinates by moving in one of 8 directions (described in section 0.1.1). In loop iteration the X and Y positions are measured using the encoders, these positions are compared to the desired position and an error is generated. This error is fed into the motor controllers which produces voltages that are applied to the motors to move the system to the required position.

0.2.2 Directional Control

Directional control has three main components, a safety VI that reads the limit switch states and receives control signals. If a directions limit switch is pressed, the control signal in that direction will not be passed onto the next sub VI closer to the motor voltage writing stage. After this, the X and Y control signals deviate to two different controllers, such that each direction can have a different voltage and hence speed, so a range of directions can be traversed. Both X and Y controllers create two voltage signals, one for each motor, these signals are then added together and sent to the motors controlling. Thus controlling the trajectory.

0.2.3 Position Tracking

Position is tracked by processing the encoder count and passing them through the equations of motions $\Delta X = \frac{\Delta A - \Delta B}{2}$ and $\Delta Y = \frac{\Delta A + \Delta B}{2}$, where ΔA and ΔB are the change in encoder counts for motor A and B respectively. By using the sums of encoder counts instead of changes, the X and Y positions can be tracked. The resulting equations are $X = \frac{A-B}{2}$ and $Y = \frac{A+B}{2}$, these are what was implemented in control program to achieve position tracking.

As the encoders provided only count up while turning both clockwise and anti-clockwise, the equations above could not be directly implemented. Our initial solution attempted to bypass the equations and simply add counts to the overall count depending on the direction of travel, however this proved both inaccurate and imprecise.

Our final solution was to take a difference of pulse counts captured at one time and a time unit before it, then to either add or subtract the difference to sum depending on the polarity of the motor voltage. In effect this behaves the same as an encoder decrementing when turning anti-clockwise and incrementing while turning clockwise. This system can be modelled by:

$$P(t + \Delta t) = sgn(M_{voltage})(\Delta P) + P(t)$$

Where $P(t)$ is the pulse count at a given time and $sgn(M_{voltage})$ is the sign function of motor voltage.

In order to account for the inertia of the system, $M_{voltage}$ was implemented to behave as a latch, effectively the $M_{voltage}$ keeps the previous sign when it returns to zero. So if the motor is running with a negative voltage and then drops to zero, the $sgn(M_{voltage})$ will be equal to -1, in order to count the degrees rotated while the the motor decelerates.

In order to convert the encoder pulses into millimetres we drew lines, 10mm in length then recorded the respective pulses in the X direction. Dividing the the line length by the pulse count resulting in a conversion factor of 46.3 pulses per millimetre. The conversion from pulses to millimetres was done by dividing the pulse count by this factor.

This method resulted in very precise and accurate tracking.

0.2.4 Moving From Position to Position

Position to position movement is achieved simply by having taking a desired co-ordinate and subtracting it by the current co-ordinate resulting in an error $Y_{desired} - Y_{measured} = Y_{error}$ and $X_{desired} - X_{measured} = X_{error}$. Depending on the sign of the error in each direction the motor control system would behave accordingly. If

Y_{error} is positive the controller will direct the system to move upwards, similarly a negative Y_{error} will direct the system to move downwards. The same logic applies for the X direction, however the sign of X_{error}directs the system to move in the right or left direction.

When the error reaches a desired value the control signal in that direction drops to 0 and no movement is desired in that direction. As the slide rail system is not ideal, each motor has different characteristics and the two rails may have varying co-efficient of friction. Because of this, providing equal voltages to both motors and producing a right signal does not result in purely horizontal movement to the right. To combat this variance we multiply one motor voltage by a factor less than 1 until the drift not related to the desired direction has reached an acceptable amount.

Ideally we would be using our speed measurement to adjust the voltages automatically until both speeds are equal, sadly we did not have enough time to implement this functionality. It would likely improve plotting accuracy by a significant margin. Our current method produces a minimum drift of 1mm in either Y direction across a 30mm X movement, the point to point control does account for this, but the resulting plot appears rippled in severe cases of drift.

0.2.5 Speed Measurement and Control

Speed or frequency of sensor feedback is measured by taking a taking the difference of a pulse count over a time period and dividing the difference by the time difference, which is modelled by this equation:

$$s(t) = \frac{\Delta P}{\Delta t} = \frac{P(t) - P(t - \Delta t)}{\Delta t}$$

Where $s(t)$ is the speed and Δt is the polling rate, how often the program calculates the speed, Δt can be made smaller than 100ms for very low speeds in order garner a more accurate reading. At $30\frac{mm}{s}$ a 100ms polling rate is adequate for an accurate reading.

Speed is controlled by adjusting the voltage going to the motors, for plotting the circle and rectangle we have set voltages at which we found the most consistent results. However the accuracy of each plot is slightly dependant on the size of the rectangle and circle, the number of points plotted in the circle and the allowable error range as well as the speed.

0.2.6 Acceleration control

Acceleration was tested out using PID control for both the rectangle and the circle, however we unable to derive adequate values for the gains such that the vertical drift while travelling horizontally was minimised. Due to the motors having non linear rotation responses to different voltages that is, the drift was at less than 1mm vertically for every 50mm travelled horizontally at 4V but as we increased the speed (voltage) the drift increased as well, leading to plot with a few small jumps.

Acceleration is implemented in the manual control panel, it takes a voltage input and increments that value to the initial speed every 100ms seconds, which is useful for calibrating the position control and moving point to point in order to start drawing a new shape.

0.3 Base Collision Prevention

Having the motor controllers receive Boolean signals from all the control tabs allowed us to easily disable directional signals corresponding to the appropriate limit switch being hit. If the program reads a control signal for the up direction and the top limit switch is hit, the Safety Box VI will prevent the control signal from being assigned a voltage. As the safety VI processes all signals going to the motor controller we have ensured that any control system we have implemented is unable to send signals to the controller that would attempt to move the system out of bounds.

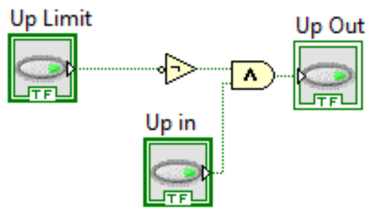


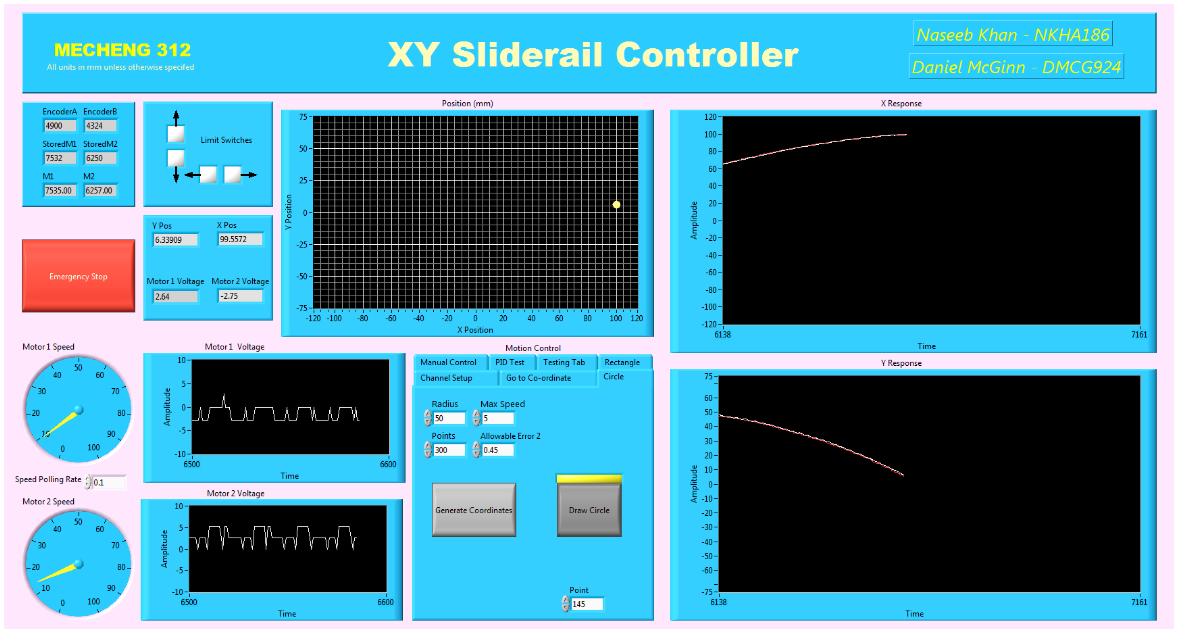
Figure 1: Implementation of Base Collision Prevention

0.4 User Interface

We designed the user interface so that the user can easily see the performance of the XY slide rail. It also displays enough information to easily debug and improve the performance of the control program even further, without having to probe objects in the back panel. We found this very useful in finding the optimal points, radius, speed and allowable error for plotting our circle.

It features a XY chart for quickly viewing the current position of the plotter, a numeric output for both X and Y positions, motor speeds and voltages. An emergency stop button that is immediately visible due to its colour contrasting with the rest of the UI. The most useful charts in the UI for performance measurement were the X and Y responses, they take up roughly half the UI and make it very clear to the user how well the plotter is tracking the position, as it displays the desired and actual position.

The colour selection creates distinction between each element of the interface, while making the important aspects stand out to the user.



EncoderA 4900
EncoderB 4324
StoredM1 7532
StoredM2 6250
M1 7535.00
M2 6257.00
Y Pos 6.33909
X Pos 99.5572
Motor 1 Voltage 2.64
Motor 2 Voltage -2.75
Speed Polling Rate 0.1
Radius 50
Max Speed 5
Points 300
Allowable Error 2 0.45
Point 145

Figure 2: User Interface

0.5 Shape Drawing

0.5.1 Rectangle

To draw the rectangle, we simply gave our position to position controller a set of four points. Our implementation resulted in consistent plotting, when a rectangle was drawn repeatedly given the same inputs the deviation from the previous rectangle was less than a millimetre. The shifting of the rectangle was only noticeable when it was drawn more than 10 times without stopping, however there were slight jumps on the bottom line due to motor drift.

The generation of co-ordinates takes width and height as inputs and calculates co-ordinates based on the position that the system is at currently, which is described in the table below.

Figure 3 shows a very accurate rectangle, however there are slight jumps in the bottom line due to Y drift while travelling in the X direction.

Point of Rectangle	X Co-Ordinate	Y Co-Ordinate
0	Initial X	Initial Y
1	Initial X	Initial Y + Height
2	Initial X + Width	Initial Y + Height
3	Initial X + Width	Initial Y
0	Initial X	Initial Y

Table 2: Co-Ordinates of Rectangle

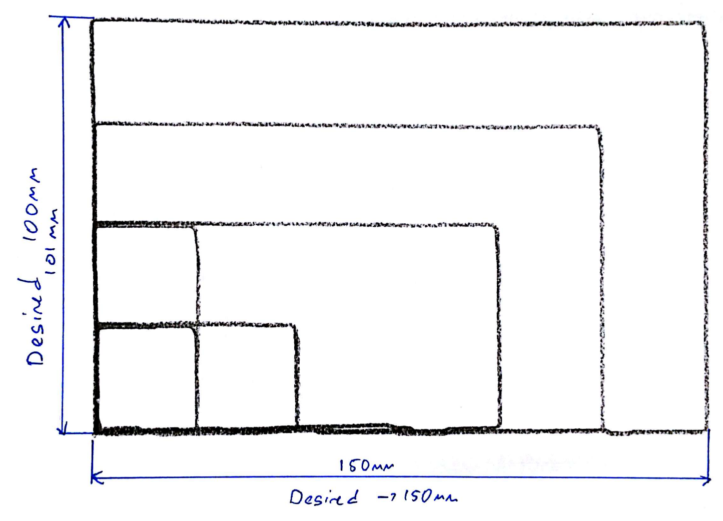


Figure 3: Drawing of Rectangles

0.5.2 Circle

The XY platform draws a circle with the input of a radius, speed, number of points, and allowable error. The platform is actually drawing a polygon with a large number of points, which effectively draws a circle. To generate these points, a mathscript is used. Once the user has input the necessary values, pressing the "Generate Coordinates" button on the front panel will prompt the code to run. The script divides 2π by the number of points to get the angle change between each point. The angle and radius are then converted from polar coordinates to Cartesian coordinates with the equations $X = r\cos(\theta)$ and $Y = r\sin(\theta)$. The circle is transposed to set the left of the circle to the origin and the initial position of the XY platform is added to each point. The outputs of the code are matrices for the x position and the y position. A sample of this code is available in the appendix with a Matlab plot of the output matrices.

A second mathscript is used to fetch the next destination from these matrices. Once the user presses the "Draw Circle" button on the front panel, the second code will send the first point to the dydx subvi, which in turn prompts the motor controller to move to the first point. Once that task is complete, the mathscript will fetch the next set of points from the matrices until all the points have been plotted. A running tally of the points that have been plotted is generated for use of the second code in fetching the next point and is also displayed for the user on the front panel.

The control system with an allowable error of 0.45mm for each point to point movement, 50mm radius and 300 points draws the circle shown on the front page. It is 100mm in the vertical direction and 97mm in the vertical direction. To improve this, equalising both motor speeds would likely increase the accuracy of the drawing.

The circle shown below is more innaccurate, we assume this was due to the motor overheating. As the thermal trip activated once during this plot. Another contributing factor was the point to point allowable error, with 300 points the 0.45mm error effectively compounds 300 times. Given that the drawn dimensions differ only by +3mm and -5mm, we are quite happy with the result.

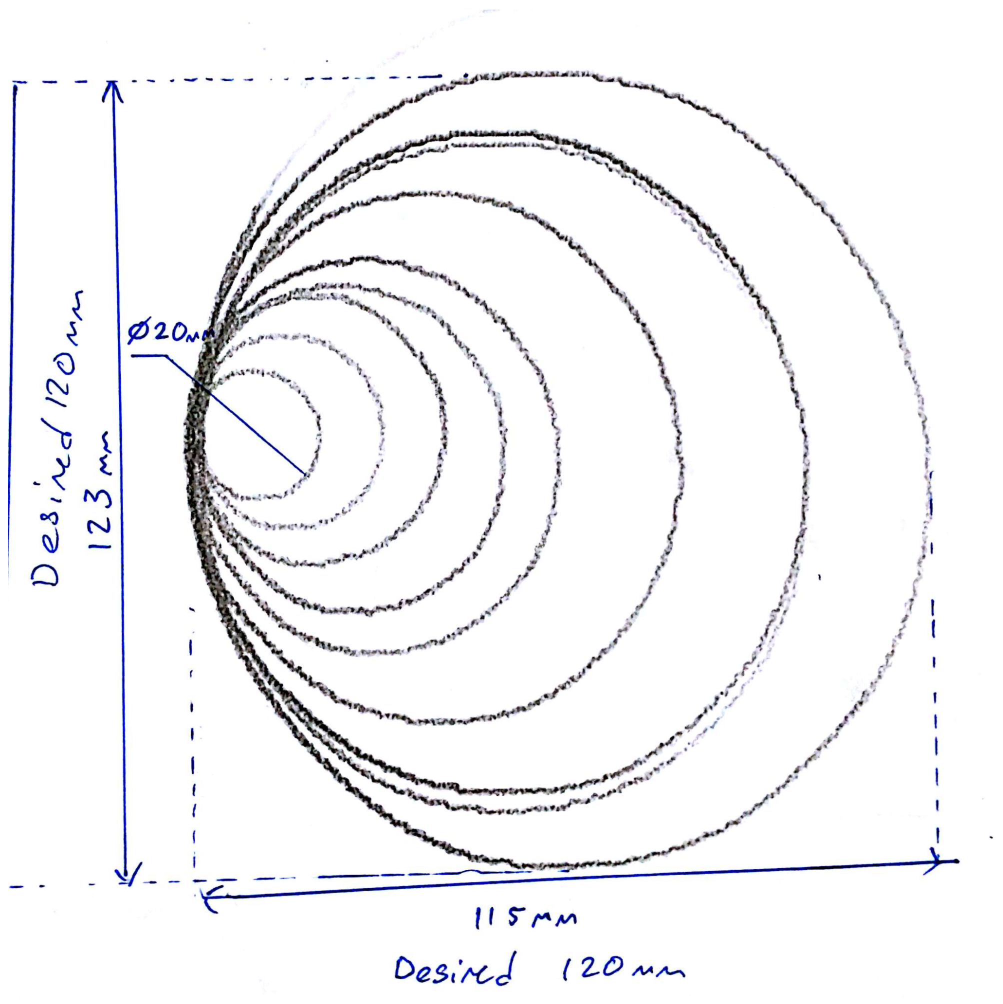


Figure 4: Drawing of Circles

0.6 Appendices

0.6.1 Matlab Code

```
% Sample MathScript to generate a circle approximation
r=3; % radius
C=[1 0]; % transpose
points=360;% number of points used to approximate the circle
Xpos = 10; Ypos = 5; % starting positions
theta=0:2*pi/points:2*pi; % the angle
m=r*[cos(theta')+C(1) sin(theta')+C(2)]; % calculate Cartesian coordinates
% transpose the circle so the left side is at the origin
xpos=m((points/2)+1:end,1); xpos((points/2)+1:points+1)=m(1:(points/2)+1,1);
% create a matrix to store the X positions in the proper order
ypos=m(:,2); % create a matrix to store the Y positions
xpos=xpos+Xpos; % add the initial position to the X positions
ypos=ypos+Ypos; % add the initial position to the Y positions
plot(Xpos,Ypos,'c*',xpos,ypos); % Plot the initial position and the circle
```

Figure 5: Matlab Code

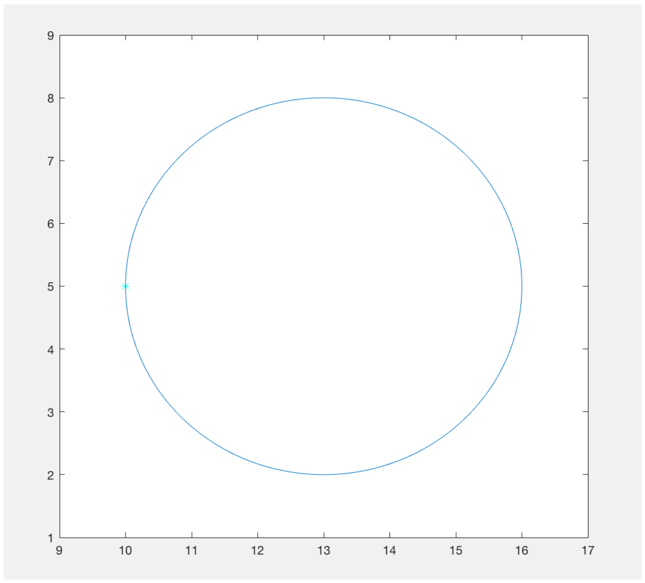


Figure 6: Matlab Plot of Circle Matrix of radius 3, starting at X=10,Y=5